J/F '79

interactive computing

PRESS REVIEW

Time-Sharing 'Messiah' Victorious at Last

By Miles Benson
Special to CW

Time-sharing is one of those things like pregnancy — you either have it or you don't.

What I mean is, when it comes to the use of time-sharing facilities for software development in a company, the company either does it or it doesn't. Rarely does a company allow programmers or projects the option of choosing between time-sharing and batch development.

Clickety-Clack and Western Railroad is one of those nontime-sharing shops. It is an unwritten company policy that "thou shalt hold no CRT images before thee, and thy testings shall only spew forth from a batch printer." As if, parenthetically, someone had said, "If God intended for people to talk to computers, he would have equipped them with modems."

And Slim Griffen knew that. When Slim hired in at Clickety-Clack five years ago and was given a tour of the computer center, his only comment had been, "But where is the terminal room?" When his manager-escort had responded "The what?", Slim's worst fears had been realized. A computing shop without time-sharing, Slim felt, was like a Cadillac in a gas crisis — it was elegant and impressive, but not of much value.

You see, Slim's computing career had been nurtured at P.Q. Aresti U., a favorite haunt of intellectual folk and a veritable bastion of belief in time-sharing. Basketweaving majors used terminals at Aresti. Janitorial services used a computerized manpower allocation system. And even the football coach played simulated games via a Decwriter in the locker room.

With that background, you may wonder why Slim ever accepted the offer from Clickety-Clack. Well, there are many perversities in human nature. Perhaps the worst of them is a missionary attitude. Slim really believed God was on *his* side and that time-sharing would eventually become a way of life on his new job. He even saw himself as the Messiah on the White Steed (his training in comparative religion was not up to the standards of his computer science, as you can see) who would ride in and slay the dragon of batch printers.

It was also true that he deeply wanted to live in the Monica Beach area along the Northern California coast, and Clickety-Clack was really the only computing game in town. But that practical explanation lacked romance, and Slim tried not to think of it.

The Messiah biz at Clickety-Clack was frustrating, Slim found. No, not just frustrating — it was impossible. The power structure surrounding Slim hadn't the slightest interest in Slim's terminal-in-every-pot plot. And several years went by in which his White Steed stood around and swished flies with its tail, ate hay and did all the other things White Steeds are wont to do. But it never came near the batch printers.

But Slim was not a quitter. He still knew he was right. And if logic would not move the establishment, then perhaps being devious would.

Now it chanced that Clickety-Clack was involved in a peculiar business at the same moments in time in which Slim's devious plot was conceived. Although most of the software developed at Clickety-Clack was for internal consumption only ("sticking to our own tracks" was the mildly humorous in-plant expression), a decision had been made to develop some "end-product" software.

In the course of sticking to its own tracks, Clickety-Clack had built a fairly nifty train scheduling algorithm. When some of the other railroads began making inquiries about the system built on that algorithm (it was called Train Automated Routing System, or Tars), Clickety-Clack computing management quickly realized it had a potential

money-maker on its hands. Its response to the other railroads was a brief description of the system and a promise that a proposal to define and implement a generalized Tars system for multirailroad use would be forthcoming.

And Slim was assigned to the writing of that proposal. The plot thickens.

A 'Natural'

Slim was a natural to work on the Tars proposal. He had designed most of the system and implemented a fair chunk. And what he hadn't built himself, he had learned inside-out. If anyone knew the Clickety-Clack-specific parts of Tars, the parts that needed redoing, it was Slim.

Well, to get the story moving again, Slim dashed off a technically credible, well-worded, tactically sound proposal in record time. It was when an audit committee was assigned to go over Slim's work before the proposal went out that the thickening plot really almost clotted.

I was assigned to that audit committee — along with two Clickety-Clack computing managers. I was asked to look at the technical details. The managers, naturally, looked at the Big Picture.

Slim had done a superb job. The managers were pleased with the Big Picture. The technical details were impeccable, And it was only on my second reading that I came to realize what Slim had done.

Buried deep in the proposal, in a section labeled "Implementation Methodology," was a pass-over-quickly sentence that said, "Development of the generalized Tars system shall be done in a structured manner, via a modular design and utilizing time-sharing techniques."

On first reading, I had zipped past it as Proposalized Motherhood. If I had quit then, no one else would have ever noticed it. Managers seldom read sections which begin with the word "implementation" or end with "methodology." That level of detail they generally find boring. And, of course, Slim knew that.

Have you got the picture here? Slim had slipped into a formal Clickety-Clack document a promise to use time-sharing. If it went unchallenged and a contract resulted from the proposal, Clickety-Clack's unwritten code *had* to change. Slim, mounted once again on his White Steed, had found a back-door approach to waging his fervent religious war.

And I was standing in his path. It was my responsibility, as part of the audit team, to report my findings. But — and here is where the plot really crystallizes — there was a loophole. Whereas the managers reported their findings to management, I needed only report my findings to the technologist — Slim.

I'd always been somewhat sympathetic to Slim's battle. I'm not sold on time-sharing as a way of life, mind you, but I'd like to see it tried in order to have more data for a better decision.

You see, I love the idea that management's disinterest in the details of computing technology can be used in potent ways against it. I think I slid Slim's devious plot through for that reason more than any other.

Members: I find it hard to agree with this author's conclusions, but I think we should be aware of his arguments. *HS*

Blows Against the Timesharing Empire

BY JOHN WALKER

The development of microcomputers has, recapitulated the development of large computers of the Fifties and the minicomputers of the Sixties. The first systems had tiny amounts of memory; were operated at the bit level with switches and lights; and were built and used mostly by people with intimate knowledge of hardware. As the price of memory declined and an infrastructure of software was built, systems advanced to the point where they could be used by software-oriented people, programming in either BASIC or assembly languages. The past year has seen the introduction of complete application packages for microcomputers and turnkey microcomputer systems.

As the declining price of semiconductors continues its asymptotic skid towards zero, the products of ingenuity from peripheral designers seem never to stop. In a couple of years one may be able to assemble a system comparable to contemporary medium-scale mainframes for a price not much more than today's personal computers. Wonderful, but what do we do with it? Obviously, because we're repeating history so faithfully, we build a timesharing system. This is a pretty safe prediction (it being a lot easier to predict the present than the future) because at least three microcomputer manufacturers are offering timesharing systems already, and undoubtedly many more will be thundering to the finish line. This is a terrible mistake.

To understand why timesharing and microcomputers should not be mixed, we must look at the history of timesharing and the reasons behind its development and current wide acceptance in the computer community. From almost inception, computers, it was realized, could be used most effectively in a "hands-on" fast-response environment. Enormous costs of early computers, however, compelled owners to convert all available time to productive use. Rather than have a multi-megabuck computer sit idle while a programmer scratched his head, jobs were assembled in batches and run one after another with no allowances for human interactions.

The expensive components in early computers were the processor (CPU) and memory. It soon became obvious that many programs were being hindered by slow Input/Output devices. It also became obvious that the capacity of the expensive components was being wasted. To alleviate this problem, multiprogramming systems were developed which allowed several programs to run simultaneously in one computer. When one program had to wait for I/O to complete, the processor could be used to handle another program. Multiprogramming required more memory than purely serial execution, but because it used tremendously expensive CPU more heavily and did more work in a system of slightly higher cost, it paid for itself.

> **One programmer put his fist through a keyboard after the computer didn't respond for several minutes.**

Multiprogramming, which originally shared the CPU among programs sharing memory, could be extended to share memory among programs which could not all fit into memory simultaneously. With fast disk of drum storage available, a program encountering a substantial delay could be written out and another program brought in to use the expensive memory and CPU. This swapping of programs allowed delays longer than a simple I/O wait to be accomodated without tying up expensive memory.

With continued refinement of multiprogramming, a solution was found for providing "hands-on" computing without dedicating an horrendously expensive computer to each user. If enough users could be collected together and connected to one computer, whenever one user was thinking, typing-in, or receiving a response from the computer, that machine could process work submitted by another user and thus keep the CPU busy. Because CPU speed is so great and almost no processor time is needed to type out information on the user terminal or receive characters from a keyboard, each user generates a minimal amount of time over a long period. The timesharing user tends to request short bursts of computing separated by waiting-times measured in seconds. Hence, in theory, the user will not notice that he is sharing a computer with many people. He will have the illusion that he is the master of a dedicated computer, ready to respond to his demands. Finally, by inducing each user of a timesharing system to pay a relatively low price user fee, say $20/hour, the high cost of the system can be recovered.

We see, then, that the driving force that created timesharing was economic; finding a way to provide many users with interactive computing even though computers and memory were very costly. Once established, a timesharing utility can provide many useful services. Since many users share a common system, they also can share programs and data files.

Now the same benefits are being offered microcomputer users with the advent of microcomputer timesharing systems. However, there is no need to go to timesharing to reap its benefits. In effect, we must change our attitude about computers, we must learn to view communication, not processing, as the key. Timesharing, as already defined, is the sharing of central processors and memory among a number of users so that that all users do not simultaneously receive service. With the price of processors and memory dropping, the practice that made sense in the Sixties is stupid in the Seventies. Timesharing is widely identified with benefits that evolve from its adoption on large computers. However, because the U.S. proceeded from stagecoaches to trains to airplanes for long distance travel doesn't mean a developing country today has to go

through the same steps. Microcomputer users should pause and think before following the path big system people took into timesharing.

What's so bad about timesharing, anyway? Let's take a look at the technical problems that plague timesharing. The most serious problem stems from the fact that timesharing is, after all, an illusion. As long as people "play by the rules" and interact on a line-by-line basis with little computation, everything works fine. Before long, somebody will ask for a substantial amount of computation, such as asking the computer to determine whether 2,363,383,927,257 is prime. Because you expect this procedure to take a while, you're not upset when the answer doesn't come back right away, and because your computation can "mop up" time not being used by other people, all is well. Suppose, however, that somebody else has a burning desire to know the 987th digit of pi right at the same time you started your program. Now we have two mops contending for one CPU, and somebody is going to lose. If the time is equally split, your program will now take twice as long to run. And what about you? You're sitting there fuming at the terminal wondering why what took a minute yesterday takes two (or five or ten or twenty) today. The concept of timesharing finds its own negation in the following slightly cynical "Laws of Timesharing" (with apologies to C. Northcote Parkinson):

1. Computations grow in complexity to fill existing processor power.

2. Programs grow in size to fill existing memory.

If these two widely-observed rules hold, the self-aggrandizement of programs in a timesharing system will eventually lead to inter-program contention that degrades response time. As a result:

3. Timesharing is characterized by unpredictable variation in response time.

The unpredictability of response time is the most frustrating aspect of timesharing. Large systems can be beautiful for obtaining a fast answer to a big problem. But the system can be singularly infuriating when it pauses for up to a minute when asked to do something like "list my five line program" that a microcomputer will always do instantly. (Long unpredictable delays make some people positively homicidal. I know one programmer who put his fist through a terminal keyboard after waiting several minutes for a system to respond.)

Another problem in timesharing is the catastrophic impact of a system crash or breach of security on its users. As systems grow more and more complex, there are more hardware components to break and more dark corners of software in which bugs may lurk. As a result, large timesharing systems tend to be less reliable than small dedicated systems. Besides, they are hellishly difficult to maintain and enhance. When the system fails, many users are simultaneously infuriated. If the users are all in the same location, mobs dangerous to the health and welfare of systems programmers can form. When many users share a system, they may also share programs and data. Unfortunately, unless the system is very careful, they may also steal or destroy each other's programs or data. Millions of dollars of Government money have been used to develop systems that do not have these problems. But no system exists that allows both cooperation and privacy. Some very good systems have been developed, but none of them has remained secure against pilferage attempts of clever programmers.

Microcomputers now in use make timesharing obsolete...

The microcomputers we now use have made timesharing obsolete. It is absolutely ludicrous to have four users sharing a computer that costs less than the terminal each one is using to access the system. We should take advantage of our position in the "third invention" of computers and leapfrog timesharing to achieve its lofty goals. If we develop networks to allow microcomputers to intercommunicate, we can then share programs and data without encountering the infuriating problems of timesharing. We must view communication as the dominant aspect of a computer's function, and build software and hardware structures based on this concept. Some systems embodying this concept have already been built and may serve as a prototype for future developments.

By proper application of networking, we can build systems that provide computers to perform most tasks rapidly, reliably and predictably. It should be a system built to the scale of a human being, not a corporation. Furthermore, it should be a system that a person can understand, can use effectively, and can maintain without managers, supervisors, inter-office memoranda, PERT charts, or a horde of narrow specialists. Such a system will be a network of intelligent components, and will intercommunicate with each other to allow the sharing and communication provided by current central timesharing systems. Because other users will be "at arm's length" over a network, it will be easier to build systems that are probably secure compared to complex centralized systems where interfaces between programs are numerous and complex. Most user tasks can be done locally, on the user's own machine. Only when communication is required need a request be made that could cause unwanted delay.

Changing economics of computer hardware now have solutions to problems that timesharing never solved. Small system people like ourselves have the outlook, experience and independence to build our own systems. These special designs will return computers to the people they were built to help in the first place. It has been said that computer software systems tend to reflect the structure of the organization that produces them. Is it little wonder that most existing systems are centralized and hierachical. Too many systems that promise "distributed processing" deliver, instead, a more complex terminal connected to an even bigger central site. Microcomputing, with its thousands of basement inventors and garage-shop entrepreneurs, has the "organizational structure" to build loose, distributed, egalitarian, flexible and powerful systems. Let's do it. □

John Walker has been involved in the design, implementation, optimization and support of large-scale timesharing systems for over eight years. Recently, as a partner in Marinchip Systems, he has worked on developing a distributed network operating system based on the ideas in this article.